Journal of Computational and Applied Mathematics 30 (1990) 71–79
North-Holland

A divide and conquer method for polynomial zeros

T.L. FREEMAN
Department of Mathematics, University of Manchester, Manchester, UK M13 9PL

R.W. BRANKIN
NAG Ltd., Oxford, UK OX2 8DR

Received 24 March 1989
Revised 7 September 1989

Abstract: The problem of factorising an nth-degree polynomial $P_n(x) = x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$, where the coefficients a_i are real, into two polynomials $Q(x)$ and $R(x)$ of degrees n_Q and n_R can be posed as a system of n quadratic equations. An efficient implementation of Newton's method for the solution of these equations is achieved by exploiting the block Toeplitz structure of the Jacobian matrix which arises when n_Q and n_R are restricted to $\{m-1,\ m,\ m+1 | m = [\frac{1}{2}n]\}$, where $[x]$ is the largest integer $\leqslant x$. This polynomial factorisation permits the development of a divide and conquer method for the simultaneous calculation of all the zeros of a polynomial.

Keywords: Polynomial zeros, divide and conquer, Newton's method, Toeplitz matrix.

1. Introduction

For many years the divide and conquer approach has been used to solve successfully a variety of numerical problems. Perhaps the best-known examples of divide and conquer methods are adaptive quadrature methods in numerical integration [15, p.221] and the FFT algorithm for calculating discrete Fourier transforms [7]. A more recent application of the divide and conquer approach has been to the calculation of the eigensystem of a symmetric tridiagonal matrix [8,9,11,14,16]. Much recent interest in divide and conquer methods has been stimulated by the need to develop numerical algorithms suitable for parallel computers (see, for example, [20]).

In this paper we demonstrate how the divide and conquer approach can be used to calculate numerically all the zeros of the polynomial

$$P_n(x) = x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n, \tag{1.1}$$

where the coefficients a_i $(i = 1, 2, \ldots, n)$ are assumed to be real. Note that, without loss of generality, $P_n(x)$ is assumed to be normalised so that the coefficient of x^n is unity. In subsequent sections we present an algorithm for dividing $P_n(x)$ into two polynomials $Q(x)$ and $R(x)$, each with real coefficients and each of degree $m-1$, m or $m+1$, where $m = [\frac{1}{2}n]$ and $[x]$

denotes the largest integer $\leqslant x$. The cases when n is divisible by 4, n is odd and n is even but not divisible by 4, are treated separately in Sections 2, 3 and 4. The algorithm is based on a Newton iteration to solve the quadratic equations which relate the unknown coefficients of the polynomials $Q(x)$ and $R(x)$ to the known coefficients of $P_n(x)$. With such a Newton iteration it is necessary to solve, on each iteration, a system of linear equations with the Jacobian matrix of the quadratic equations as the coefficient matrix. In this case the Jacobian matrix is a block Toeplitz matrix so that the linear equations can be solved in $O(n^2)$ operations and the Newton iteration can be implemented very efficiently.

In Section 5 we consider the numerical performance of the proposed method on several of the test problems proposed by Henrici and Watkins [13].

2. Factorising a polynomial of degree divisible by 4

When $P_n(x)$ of (1.1) has degree divisible by 4 then there exist polynomials $Q_N(x)$ and $R_N(x)$ of degrees $N = \frac{1}{2}n$ such that

$$P_n(x) = Q_N(x)R_N(x), \tag{2.1}$$

and $Q_N(x)$ and $R_N(x)$ each have real coefficients. If we write

$$Q_N(x) = x^N + b_1x^{N-1} + \cdots + b_{N-1}x + b_N, \tag{2.2}$$

$$R_N(x) = x^N + c_1x^{N-1} + \cdots + c_{N-1}x + c_N, \tag{2.3}$$

then equating coefficients of x^k $(k = n-1, n-2, \ldots, 0)$ in (2.1) leads to the system of quadratic equations for b_i, c_i $(i = 1, 2, \ldots, N)$

$$\begin{aligned}
b_1 + c_1 - a_1 &= 0,\\
b_2 + b_1c_1 + c_2 - a_2 &= 0,\\
b_3 + b_2c_1 + b_1c_2 + c_3 - a_3 &= 0,\\
&\vdots\\
b_N + b_{N-1}c_1 + b_{N-2}c_2 + \cdots + b_1c_{N-1} + c_N - a_N &= 0,\\
b_Nc_1 + b_{N-1}c_2 + \cdots + b_1c_N - a_{N+1} &= 0,\\
b_Nc_2 + b_{N-1}c_3 + \cdots + b_2c_N - a_{N+2} &= 0,\\
&\vdots\\
b_Nc_N - a_n &= 0.
\end{aligned} \tag{2.4}$$

These equations can be written as

$$\boldsymbol{f}(\boldsymbol{b}, \boldsymbol{c}) = \boldsymbol{0}, \tag{2.5}$$

where the kth element of $\boldsymbol{f}$ is

$$f_k(\boldsymbol{b}, \boldsymbol{c}) = \phi_k(\boldsymbol{b}, \boldsymbol{c}) - a_k, \quad k = 1, 2, \ldots, n, \tag{2.6}$$

and

$$\phi_k(\boldsymbol{b}, \boldsymbol{c}) = \begin{cases} b_k + \sum_{j=1}^{k-1} b_{k-j}c_j + c_k, & k = 1, 2, \ldots, N, \\ \sum_{j=k}^{n} b_{j-N}c_{k-j+N}, & k = N+1, N+2, \ldots, n, \end{cases}$$

and where $\boldsymbol{b}^{\mathrm{T}} = (b_1, b_2, \ldots, b_N)$, $\boldsymbol{c}^{\mathrm{T}} = (c_1, c_2, \ldots, c_N)$.

The Newton iteration for the solution of (2.5) is given by

$$\begin{pmatrix} \boldsymbol{b}^{(k+1)} \\ \boldsymbol{c}^{(k+1)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}^{(k)} \\ \boldsymbol{c}^{(k)} \end{pmatrix} + \begin{pmatrix} \delta\boldsymbol{b}^{(k)} \\ \delta\boldsymbol{c}^{(k)} \end{pmatrix}, \tag{2.7}$$

where

$$\boldsymbol{J}^{(k)} \begin{pmatrix} \delta\boldsymbol{b}^{(k)} \\ \delta\boldsymbol{c}^{(k)} \end{pmatrix} = -\boldsymbol{f}^{(k)}, \tag{2.8}$$

and $\boldsymbol{J}$ denotes the Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{x}^{\mathrm{T}} = [\boldsymbol{b}^{\mathrm{T}}\boldsymbol{c}^{\mathrm{T}}]$. For $i = 1, 2, \ldots, n$,

$$J_{ij}(\boldsymbol{x}) = \begin{cases} \dfrac{\partial f_i}{\partial b_j}(\boldsymbol{x}), & j = 1, 2, \ldots, N, \\ \dfrac{\partial f_i}{\partial c_{j-N}}(\boldsymbol{x}), & j = N+1, N+2, \ldots, n. \end{cases}$$

The superscript k on $\boldsymbol{J}$ and $\boldsymbol{f}$ denotes evaluation at $\boldsymbol{x}^{(k)}$. For the function $\boldsymbol{f}$ of (2.5), $\boldsymbol{J}$ is given by

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}, \tag{2.9}$$

where

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ c_1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ c_{N-1} & \cdots & c_1 & 1 \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ b_1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ b_{N-1} & \cdots & b_1 & 1 \end{pmatrix},$$

$$\boldsymbol{C} = \begin{pmatrix} c_N & c_{N-1} & \cdots & c_1 \\ 0 & c_N & \cdots & c_2 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_N \end{pmatrix}, \qquad \boldsymbol{D} = \begin{pmatrix} b_N & b_{N-1} & \cdots & b_1 \\ 0 & b_N & \cdots & b_2 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & b_N \end{pmatrix}.$$

This is a block matrix with blocks which are Toeplitz matrices [12]. If we partition

$$\boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{pmatrix},$$

where f_1 and f_2 are N-vectors, and omit the superfix k, then the linear system (2.8) can be written as

$$A\delta b + B\delta c = -f_1, \tag{2.10}$$

$$C\delta b + D\delta c = -f_2. \tag{2.11}$$

It can be shown [4,22] that the inverse of a triangular Toeplitz matrix is a triangular Toeplitz matrix. Thus B^{-1} and D^{-1} (provided that $b_N \neq 0$) are lower and upper triangular Toeplitz matrices, respectively. If we premultiply (2.10) by B^{-1} and (2.11) by D^{-1} we obtain

$$B^{-1}A\delta b + \delta c = -B^{-1}f_1, \qquad D^{-1}C\delta b + \delta c = -D^{-1}f_2.$$

Hence eliminating δc gives

$$(B^{-1}A - D^{-1}C)\delta b = D^{-1}f_2 - B^{-1}f_1. \tag{2.12}$$

Brankin [4] shows that the product of lower (upper) triangular Toeplitz matrices is a lower (upper) triangular Toeplitz matrix. Thus (2.12) can be written as

$$T\delta b = \hat{f}, \tag{2.13}$$

where $T = B^{-1}A - D^{-1}C$ is a full Toeplitz matrix and $\hat{f} = D^{-1}f_2 - B^{-1}f_1$. Equation (2.13) can be solved using the routine TSLS from the TOEPLITZ package [1]. This routine is based on the algorithm of Trench [23] (see also [24,25]), and it requires approximately $3N^2$ operations. Bunch [6] has shown that this algorithm may be unstable if the coefficient matrix is not positive definite. In extensive testing by the first author, in which (2.13) was solved both by the TOEPLITZ package and by the NAG routine F04ATF [18] which uses Crout's factorisation with partial pivoting, this potential instability was never detected. In fact the self-correcting property of Newton's method means that the occasional poor solution of (2.13) due to instability should not affect adversely the algorithm being discussed here. The total cost of calculating δb and then δc is approximately $\frac{15}{2}N^2$ operations [4].

The determinant of J of (2.9) is the Sylvester determinant and it can be shown (see [3]) that

$$\det J = \pm \prod_{i,j=1}^{N} (\beta_i - \gamma_j),$$

where

$$Q_N(x) = x^N + b_1x^{N-1} + \cdots + b_{N-1}x + b_N = \prod_{i=1}^{N} (x - \beta_i),$$

$$R_N(x) = x^N + c_1x^{N-1} + \cdots + c_{N-1}x + c_N = \prod_{i=1}^{N} (x - \gamma_i).$$

Hence J is singular if and only if $Q_N(x)$ and $R_N(x)$ have at least one common zero.

Ortega [19, Theorem 8.1.10] establishes the local convergence (in the sense that there exists an open neighbourhood of the solution within which convergence is assured) of Newton's method provided that the Jacobian matrix at the solution is nonsingular. The theorem further establishes the quadratic rate of convergence of Newton's method under mild differentiability conditions. Thus the proposed divide and conquer method is locally convergent and has a quadratic rate of convergence provided that $Q_N(x)$ and $R_N(x)$ have no common zeros, which is assured if $P_n(x)$ has simple zeros.

If $P_n(x)$ has a k-fold multiple zero α and all k copies of α appear in one of the factored polynomials, then the proposed method remains locally convergent with a quadratic rate of convergence. If however the copies of the multiple zero α appear in both of the factored polynomials $Q_N(x)$ and $R_N(x)$, then the matrix $\boldsymbol{J}$ is singular at the solution of the equations (2.4) and the local convergence of the method is no longer guaranteed. In this latter case the Newton iteration (2.7) has a slow (linear) rate of convergence, and thus in principle the difficulty could be detected by the algorithm. At present the authors are unable to suggest a remedy for the difficulty and therefore the algorithm is really only appropriate for polynomials with simple zeros.

3. Factorising an odd degree polynomial

When the degree n of $P_n(x)$ of (1.1) is odd, then there exist polynomials $Q_N(x)$ and $R_{N+1}(x)$ of degrees N and $N+1$ respectively, where $N = [\frac{1}{2}n]$ is the largest integer less than $\frac{1}{2}n$, such that

$$P_n(x) = Q_N(x) R_{N+1}(x), \tag{3.1}$$

and $Q_N(x)$, $R_{N+1}(x)$ have real coefficients. If we write $Q_N(x)$ as in (2.2) and

$$R_{N+1}(x) = x^{N+1} + c_1 x^N + \cdots + c_N x + c_{N+1}, \tag{3.2}$$

and proceed as in Section 2, we obtain the equations

$$\boldsymbol{f}(\boldsymbol{b}, \boldsymbol{c}) = \boldsymbol{0}, \tag{3.3}$$

where

$$f_k(\boldsymbol{b}, \boldsymbol{c}) = \phi_k(\boldsymbol{b}, \boldsymbol{c}) - a_k, \quad k = 1, 2, \ldots, n,$$

and

$$\phi_k(\boldsymbol{b}, \boldsymbol{c}) = \begin{cases} b_k + \sum_{j=1}^{k-1} b_{k-j} c_j + c_k, & k = 1, 2, \ldots, N, \\ c_{N+1} + \sum_{j=N+2}^{n} b_{j-N-1} c_{2N+1-j}, & k = N+1, \\ \sum_{j=k}^{n} b_{j-N-1} c_{k-j+N+1}, & k = N+2, N+3, \ldots, n. \end{cases}$$

The Newton iteration for the solution of (3.3) is given by (2.7) and (2.8) where the Jacobian matrix is now given by

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}, \tag{3.4}$$

where

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ c_1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ c_{N-1} & \cdots & c_1 & 1 \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ b_1 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ b_{N-1} & \cdots & b_1 & 1 & 0 \end{pmatrix},$$

$$\boldsymbol{C} = \begin{pmatrix} c_N & c_{N-1} & \cdots & c_1 \\ c_{N+1} & c_N & \cdots & c_2 \\ 0 & c_{N+1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & c_N \\ 0 & \cdots & 0 & c_{N+1} \end{pmatrix}, \qquad \boldsymbol{D} = \begin{pmatrix} b_N & b_{N-1} & \cdots & b_1 & 1 \\ 0 & b_N & \ddots & & b_1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & b_N & b_{N-1} \\ 0 & \cdots & \cdots & 0 & b_N \end{pmatrix},$$

and we note that $\boldsymbol{D}$ is an $(N+1)\times(N+1)$ triangular Toeplitz matrix and $\boldsymbol{B}$, $\boldsymbol{C}$ are rectangular matrices. If we partition

$$\boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{pmatrix},$$

where $\boldsymbol{f}_1$ is an N-vector and $\boldsymbol{f}_2$ is an $(N+1)$-vector, then the linear system (2.8) can be written as

$$\boldsymbol{A}\delta\boldsymbol{b} + \boldsymbol{B}\delta\boldsymbol{c} = -\boldsymbol{f}_1, \tag{3.5}$$

$$\boldsymbol{C}\delta\boldsymbol{b} + \boldsymbol{D}\delta\boldsymbol{c} = -\boldsymbol{f}_2. \tag{3.6}$$

If we let $\boldsymbol{B} = (\bar{\boldsymbol{B}} \,|\, \boldsymbol{0})$, where $\boldsymbol{0}$ is an N-vector of zeros, and let $\delta\boldsymbol{c}^{\mathrm{T}} = (\overline{\delta\boldsymbol{c}^{\mathrm{T}}}, \delta c_{N+1})$, then (3.5) can be written as

$$\boldsymbol{A}\delta\boldsymbol{b} + \bar{\boldsymbol{B}}\overline{\delta\boldsymbol{c}} = -\boldsymbol{f}_1. \tag{3.7}$$

$\bar{\boldsymbol{B}}^{-1}$ and $\boldsymbol{D}^{-1}$ (provided that $b_N \neq 0$) are lower and upper triangular Toeplitz matrices, respectively. If we premultiply (3.5) by $\bar{\boldsymbol{B}}^{-1}$ and (3.6) by $\boldsymbol{D}^{-1}$ we obtain

$$\bar{\boldsymbol{B}}^{-1}\boldsymbol{A}\delta\boldsymbol{b} + \overline{\delta\boldsymbol{c}} = -\bar{\boldsymbol{B}}^{-1}\boldsymbol{f}_1, \tag{3.8}$$

$$\boldsymbol{D}^{-1}\boldsymbol{C}\delta\boldsymbol{b} + \delta\boldsymbol{c} = -\boldsymbol{D}^{-1}\boldsymbol{f}_2. \tag{3.9}$$

$\bar{\boldsymbol{B}}^{-1}\boldsymbol{A}$ is an $N\times N$ lower triangular Toeplitz matrix and $\boldsymbol{D}^{-1}\boldsymbol{C}$ is an $(N+1)\times N$ upper Hessenberg Toeplitz matrix. If we let $\boldsymbol{E}$ denote the first N rows of $\boldsymbol{D}^{-1}\boldsymbol{C}$ and $\boldsymbol{g}$ denote the first N entries of $-\boldsymbol{D}^{-1}\boldsymbol{f}_2$, then we can eliminate $\delta\boldsymbol{c}$ from (3.8), (3.9) to give

$$\boldsymbol{T}\delta\boldsymbol{b} = \hat{\boldsymbol{f}}, \tag{3.10}$$

where $\boldsymbol{T} = \bar{\boldsymbol{B}}^{-1}\boldsymbol{A} - \boldsymbol{E}$ is a full Toeplitz matrix and $\hat{\boldsymbol{f}} = \boldsymbol{g} - \bar{\boldsymbol{B}}^{-1}\boldsymbol{f}_1$. Equation (3.10) can now be solved using the routing TSLS from the TOEPLITZ package and the overall cost of finding $\delta\boldsymbol{b}$ and $\delta\boldsymbol{c}$ is again approximately $\frac{15}{2}N^2$ operations. The determinant of $\boldsymbol{J}$ of (3.4) can be shown to be nonsingular if and only if $Q_N(x)$ and $R_{N+1}(x)$ have no common zeros.

4. Factorising a polynomial whose degree is even but not divisible by 4

In this case it can be shown that $P_n(x)$ can be factorised as

$$P_n(x) = Q_N(x)R_{N+2}(x), \tag{4.1}$$

where $Q_N(x)$, $R_{N+2}(x)$ are polynomials with real coefficients of degrees N and $N+2$ respectively, and where $N = \frac{1}{2}n - 1$. Rather than repeat the algebra of Sections 2 and 3, we just quote the form of the Jacobian matrix in this case and note that the Newton correction vector can be

calculated in approximately $\frac{15}{2}N^2$ operations as before. Assuming that $Q_N(x)$ is given by (2.2) and

$$R_{N+2}(x) = x^{N+2} + c_1 x^{N+1} + \cdots + c_{N+1}x + c_{N+2}, \tag{4.2}$$

the Jacobian matrix is given by

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}, \tag{4.3}$$

where

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ c_1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ c_{N-1} & \cdots & c_1 & 1 \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 \\ b_1 & 1 & \ddots & & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ b_{N-1} & \cdots & b_1 & 1 & 0 & 0 \end{pmatrix},$$

$$\boldsymbol{C} = \begin{pmatrix} c_N & c_{N-1} & \cdots & c_1 \\ c_{N+1} & c_N & \cdots & c_2 \\ c_{N+2} & c_{N+1} & & \vdots \\ 0 & c_{N+2} & \ddots & c_N \\ \vdots & \ddots & \ddots & c_{N+1} \\ 0 & \cdots & 0 & c_{N+2} \end{pmatrix}, \qquad \boldsymbol{D} = \begin{pmatrix} b_N & b_{N-1} & \cdots & b_1 & 1 & 0 \\ 0 & b_N & \cdots & b_2 & b_1 & 1 \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \\ & & & \ddots & b_N & b_{N-1} \\ 0 & \cdots & \cdots & \cdots & 0 & b_N \end{pmatrix}.$$

5. Numerical performance of the proposed method

Clearly a divide and conquer method for finding polynomial zeros will use the algorithms described in Sections 2, 3 and 4 to reduce a given polynomial to its simple factors (linear, quadratic and, perhaps, cubic factors) so that the zeros can be found. Two aspects of the algorithm which have yet to be described are the convergence criterion and the selection of starting values for the Newton iteration. The convergence condition for the current implementation of the algorithm is that the largest relative correction to the components of $\boldsymbol{b}$ and $\boldsymbol{c}$ is less than or equal to TOL. This test is modified to an absolute test for those components of $\boldsymbol{b}$ and $\boldsymbol{c}$ which are close to zero.

To generate the starting values we first calculate a bound r on the moduli of the zeros of $P_n(x)$ using [17, Theorem 27.1]. The $\boldsymbol{b}$ starting values are given by

$$b_i = 0, \quad i = 1, 2, \ldots, N-1, \qquad b_N = r,$$

so that initially the zeros of $Q_N(x)$ are $(r^{1/N} \times N)$th roots of unity.

The $\boldsymbol{c}$ starting values can be calculated either by forward substitution through the leading equations of (2.4) or by backward substitution through the final equations of (2.4).

We now compare the performance of this algorithm, referred to as DIVCON, with an implementation of the Durand–Kerner algorithm (POLSOL) for polynomials with real coefficients. POLSOL is based on the generalised Durand–Kerner method presented in [10] and also simultaneously approximates all the zeros of $P_n(x)$.

Table 1
Numerical performance of DIVCON and POLSOL

Test polynomial	Degree	Number of iterations	
		DIVCON	POLSOL
Brodlie [5]	6	10	12
Bairstow [2]	8	14	18
Henrici, Table 1, Problem 27	10	10	19
Henrici, Table 1, Problem 28	12	10	45
Henrici, Table 1, Problem 29	13	11	58
Henrici, Table 1, Problem 30	15	12	22
Henrici, Table 1, Problem 31	18	16	25
Henrici, Table 1, Problem 32	19	10	26
Henrici, Table 2, Problem 5	8	7	10
Henrici, Table 2, Problem 6	8	*	19
Henrici, Table 2, Problem 7	8	*	34
Henrici, Table 2, Problem 8	8	23	39
Henrici, Table 2, Problem 9	9	17	26
Henrici, Table 2, Problem 10	9	11	11
Henrici, Table 2, Problem 11	15	*	36
Henrici, Table 2, Problem 12	18	13	42
Henrici, Table 2, Problem 13	36	19	22

Table 1 summarises the performances of the two algorithms on the test problems suggested by Bairstow [2] and by Brodlie [5] together with all the Henrici test problems which have degrees greater than or equal to 8 (see [13,21]). The results were obtained on an Amdahl 5890-300 at UMRCC using about 14 decimal digit accuracy, with TOL equal to 10^{-6}. The number of iterations for DIVCON is weighted to reflect that the degree of $P_n(x)$ decreases as the algorithm progresses. Each weighed iteration of DIVCON requires approximately $\frac{15}{8}n^2$ floating-point operations and each iteration of POLSOL requires approximately $\frac{5}{2}n^2$ floating-point operations. Thus a measure of the cost of solving a given test problem by DIVCON is $\frac{15}{8} \times$ number of iterations whilst a measure of the cost of solving a given problem by POLSOL is $\frac{5}{2} \times$ number of iterations. * indicates that DIVCON fails to converge in $10 \times$ degree iterations.

Excluding those three cases where it fails to converge, DIVCON requires, on average, about half the number of iterations for convergence required by POLSOL. Additionally each iteration of DIVCON requires less (by a factor of $\frac{3}{4}$) floating-point operations compared with POLSOL. The increased efficiency of DIVCON is somewhat counterbalanced by the greater robustness of POLSOL.

Since both DIVCON and POLSOL simultaneously approximate all the zeros of a polynomial, they are very suitable for implementation on MIMD computers.

Acknowledgements

We are grateful to Professor Steven Barnett for pointing out the result on the Sylvester determinant and to Professor Ian Gladwell for valuable comments on the manuscript.

References

[1] O.B. Arushanian, M.K. Samarin, V.V. Voevodin, E.E. Tyrtyshnikov, B.S. Garbow, J.M. Boyle, W.R. Cowell and K.W. Dritz, The TOEPLITZ package user's guide, Report ANL-83-16, Argonne National Laboratory, Argonne, IL, 1983.

[2] L. Bairstow, The solution of algebraic equations with numerical coefficients in the case where several pairs of complex roots exist, Advisory Committee for Aeronautics, Technical Report for 1914–15, 239–252.

[3] S. Barnett, *Matrices in Control Theory* (Krieger, Malabar, FL, 1984).

[4] R.W. Brankin, A method for finding all the roots of a polynomial with real coefficients, M.Sc. Dissertation, University of Manchester, 1984.

[5] K.W. Brodlie, On Bairstow's method for the solution of polynomial equations, *Math. Comp.* **29** (1975) 815–826.

[6] J.R. Bunch, Stability of methods for solving Toeplitz systems of equations, *SIAM J. Sci. Statist. Comput.* **6** (1985) 349–364.

[7] J.W. Cooley and J.W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Math. Comp.* **19** (1965) 297–301.

[8] J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.* **36** (1981) 177–195.

[9] J.J. Dongarra and D.C. Sorensen, A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM J. Sci. Statist. Comput.* **8** (1987) 139–154.

[10] T.L. Freeman, A method for computing all the zeros of a polynomial with real coefficients, *BIT* **19** (1979) 321–333.

[11] D. Gill and E. Tadmor, An $O(N^2)$ method for computing the eigensystem of $N \times N$ symmetric tridiagonal matrices by the divide and conquer approach, ICASE Report No. 88-19, Langley Research Center, Hampton, VA, 1988.

[12] G.H. Golub and C.F. Van Loan, *Matrix Computations* (North Oxford Academic, Oxford, 1983).

[13] P. Henrici and B.O. Watkins, Finding zeros of a polynomial by the Q-D algorithm, *Comm. ACM* **8** (1965) 570–574.

[14] I.C.F. Ipsen and E.R. Jessup, Solving the symmetric tridiagonal eigenvalue problem on the Hypercube, Yale University Research Report YALEU/DCS/RR-548, 1987.

[15] R.L. Johnston, *Numerical Methods: A Software Approach* (Wiley, New York, 1982).

[16] A.S. Krishnakumar and M. Morf, Eigenvalues of a symmetric tridiagonal matrix: a divide-and-conquer approach, *Numer. Math.* **48** (1986) 349–368.

[17] M. Marden, *Geometry of Polynomials* (Amer. Mathematical Soc., Providence, RI, 1966).

[18] NAG, NAG Library Manual, Mark 12, NAG Ltd., Oxford, 1987.

[19] J.M. Ortega, *Numerical Analysis. A Second Course* (Academic Press, New York, 1972).

[20] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers* (McGraw-Hill, New York, 1987).

[21] R.F. Thomas, Corrections to numerical data on Q-D algorithm, *Comm. ACM* **9** (1966) 322–323.

[22] J.F. Traub, Associated polynomials and uniform methods for the solution of linear problems, *SIAM Rev.* **8** (1966) 277–307.

[23] W.F. Trench, An algorithm for the inversion of finite Toeplitz matrices, *J. SIAM* **12** (1964) 512–522.

[24] S. Zohar, Toeplitz matrix inversion: the algorithm of W.F. Trench, *J. Assoc. Comput. Mach.* **16** (1969) 592–601.

[25] S. Zohar, The solution of a Toeplitz set of linear equations, *J. Assoc. Comput. Mach.* **21** (1974) 272–276.